Interacting With Database

Mr. Nilesh Vishwasrao Patil
Government Polytechnic Ahmednagar

Specific Objectives

- To create database driven business applications using the database API'S two tier and three tier models and the Java.Sql package

Database Drivers

- Think of a database as just another device connected to your computer
- Like other devices it has a driver program to relieves you of having to do low level programming to use the database
- The driver provides you with a high level API to the database

ODBC

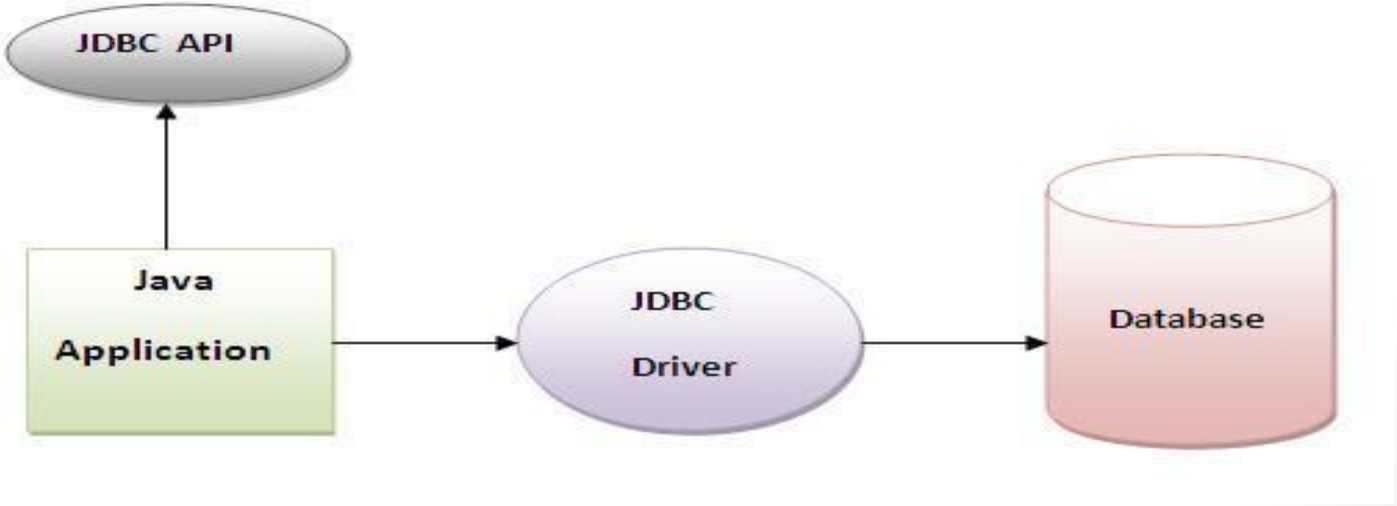
- Open Data Base Connectivity
- Developed by Microsoft for the Windows platform as the way for Windows applications to access Microsoft databases (SQL Server, FoxPro, Access)
- Has become an industry standard
- Most data base vendors supply native, ODBC, and JDBC drivers for their data base products .
- It was developed in C programming.
- It is low level, high performance interface that is designed specially for relational data stores.

JDBC

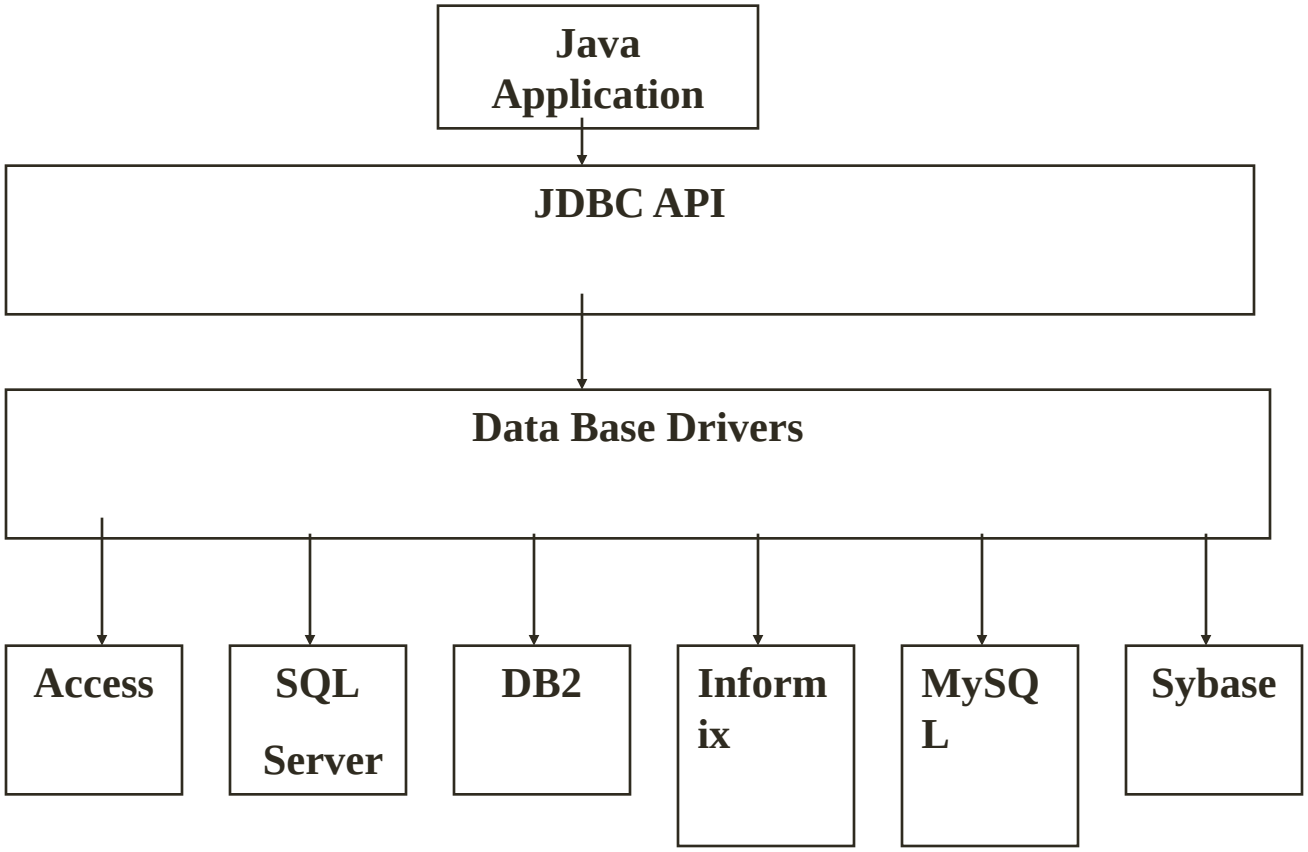
- JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server.
- Package: java.sql
- JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

JDBC API

- JDBC is a Java API that is used to connect and execute query for the database.
- JDBC API uses jdbc drivers to connects to the database.



JDBC Architecture



JDBC Driver Types

- JDBC driver implementation vary because of wide variety of OS, hardware, databases in which Java operates. It divides into 4 types:

- Type 1
  - JDBC-ODBC Bridge
- Type 2
  - Native API, partially java
- Type 3
  - JDBC Network Driver, partially java
- Type 4
  - 100% Java

Type 1 Drivers

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.
- The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

Type 1 Driver (cont.)

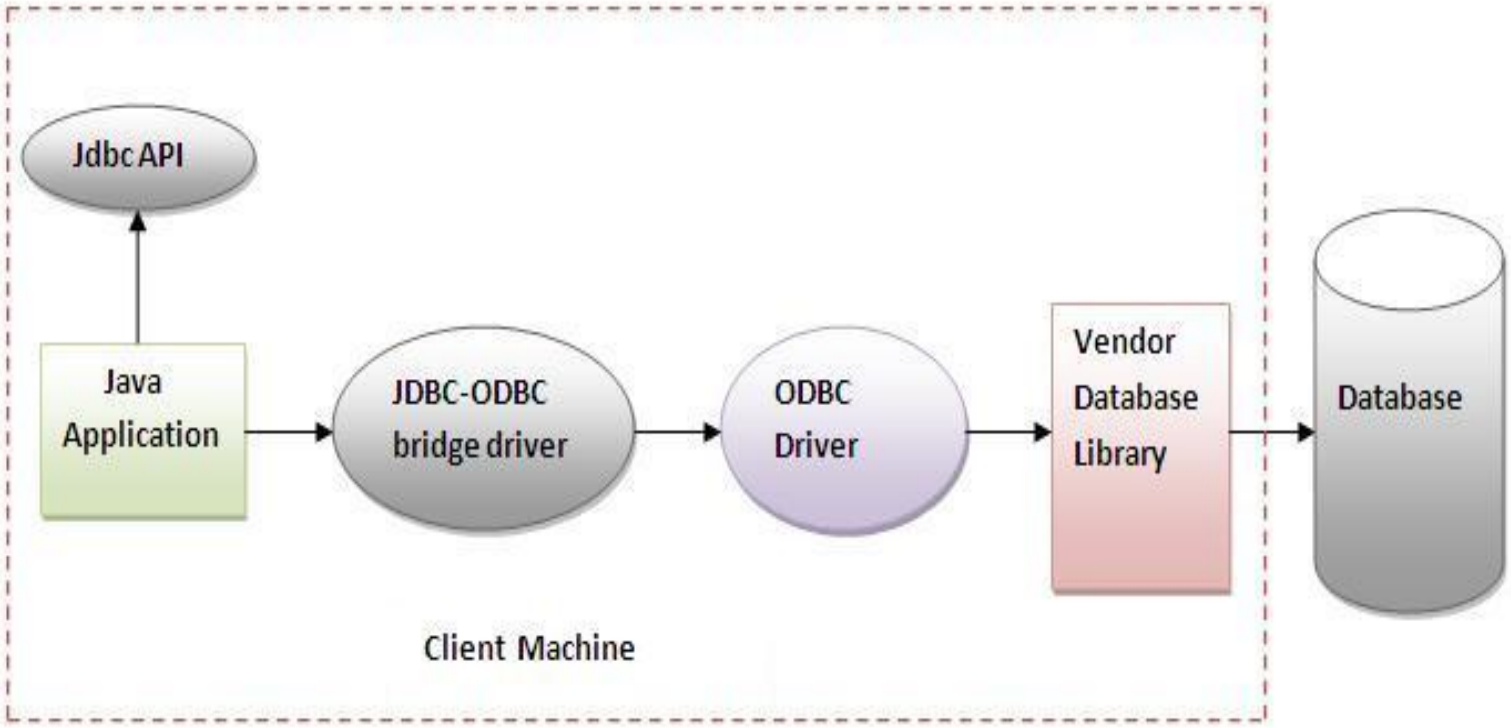


Figure- JDBC-ODBC Bridge Driver

Type 1 Drivers

- **Advantages:**
  - easy to use.
  - can be easily connected to any database.
- **Disadvantages:**
  - Performance degraded because JDBC method call is converted into the ODBC function calls.
  - The ODBC driver needs to be installed on the client machine.

Type 2 Drivers

- The Native API driver uses the client-side libraries of the database.
- The driver converts JDBC method calls into native (c/c++ API) calls of the database API.
- If we change database, we have to change native API.
- It is not written entirely in java.

Type 2 Drivers (cont.)

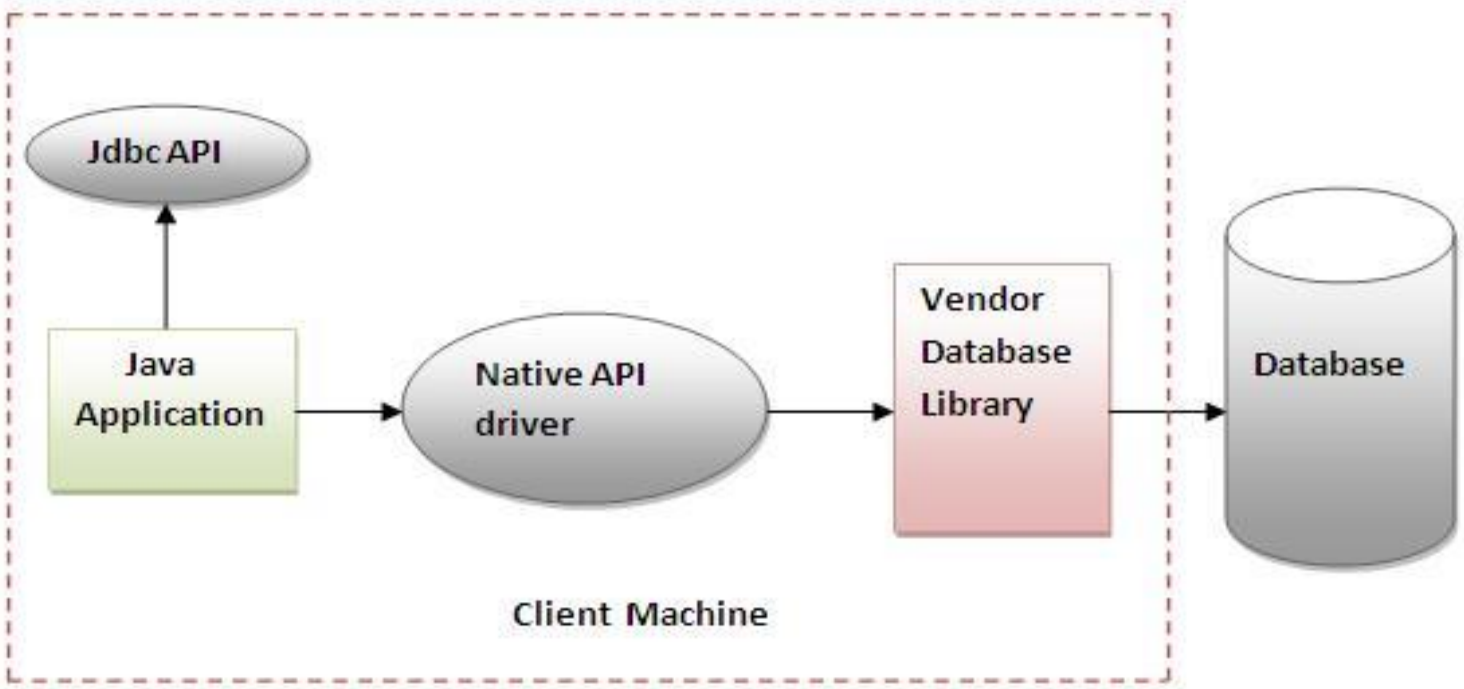


Figure- Native API Driver

Type 2 Drivers

- **Advantage:**
  - performance upgraded than JDBC-ODBC bridge driver.
- **Disadvantage:**
  - The Native driver needs to be installed on the each client machine.
  - The Vendor client library needs to be installed on client machine.

Type 3 Drivers

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.

- It is fully written in java.

Type 3 Drivers (cont.)

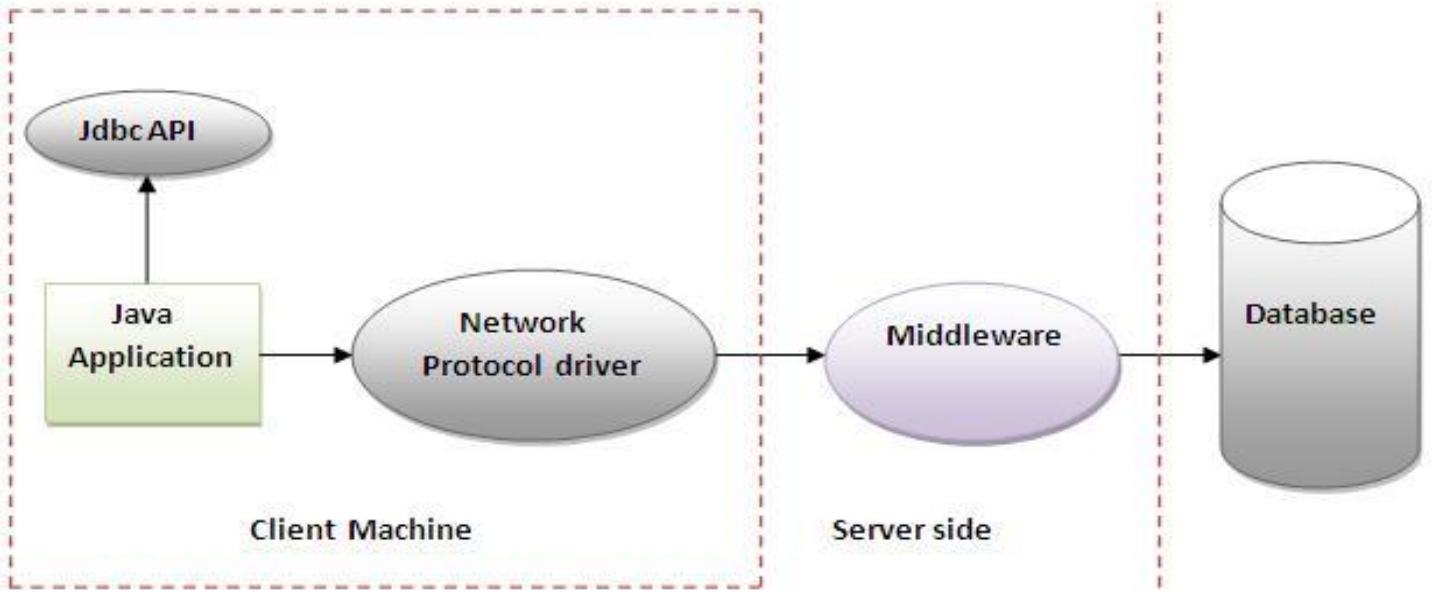


Figure- Network Protocol Driver

Type 3 Drivers

- **Advantage:**
  - No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- **Disadvantages:**
  - Network support is required on client machine.
  - Requires database-specific coding to be done in the middle tier.
  - Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type 4 Drivers

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.

- It is fully written in Java language.

Type 4 Drivers (cont.)

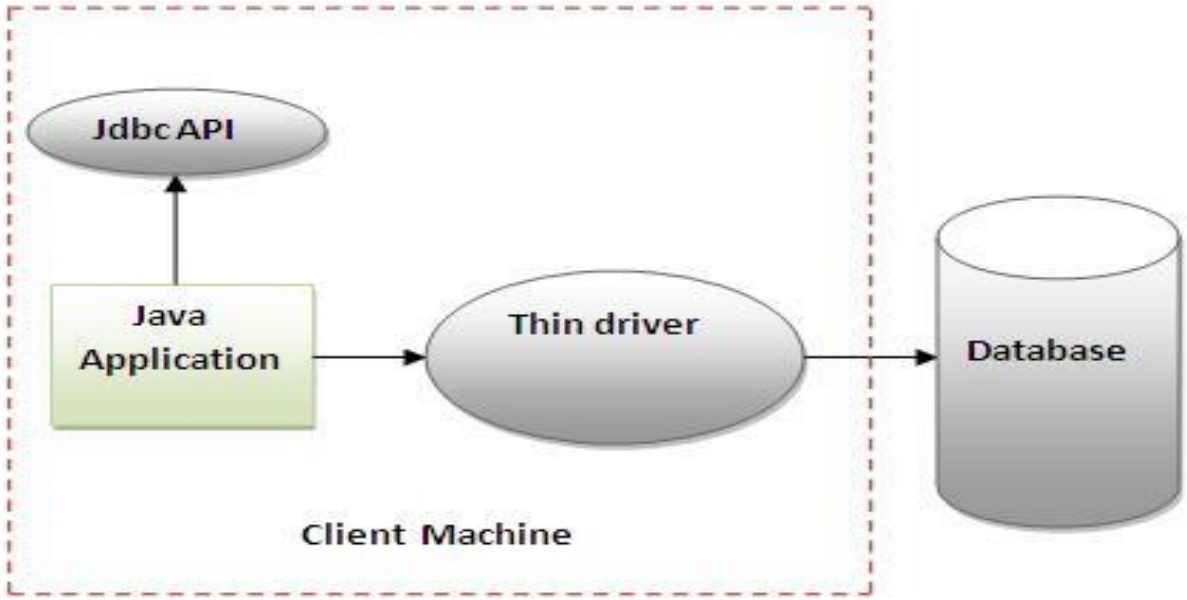


Figure- Thin Driver

Driver 4 Type

- **Advantage:**
  - Better performance than all other drivers.
  - No software is required at client side or server side.
- **Disadvantage:**
  - Drivers depends on the Database.

Which Driver should I used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing prposes only.

Steps to connect Database

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

Step 1

- The forName() method of Class class is used to register the driver class.
- This method is used to dynamically load the driver class.
- **public static void** forName(String className)**throws** ClassNotFoundException
- Example
  - Class.forName("oracle.jdbc.driver.OracleDriver");

Step 2

- The getConnection() method of DriverManager class is used to establish connection with the database.
- Syntax:
  - 1) **public static** Connection getConnection(String url)**th rows** SQLException
  - 2) **public static** Connection getConnection(String url,St ring name,String password) **throws** SQLException
- Example
  - Connection con=DriverManager.getConnection("jdbc:o racle:thin:@localhost:1521:xe","system","password");

Step 3

- The createStatement() method of Connection interface is used to create statement.
- The object of statement is responsible to execute queries with the database.
- Syntax:
  - **public** Statement createStatement()**throws** SQLExc eption
- Ex:
  - Statement stmt=con.createStatement();

Step 4

- The executeQuery() method of Statement interface is used to execute queries to the database.
- This method returns the object of ResultSet that can be used to get all the records of a table.
- Syntax:
  - **public** ResultSet executeQuery(String sql)**throws** SQLEx ception
- Ex:

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Step 5

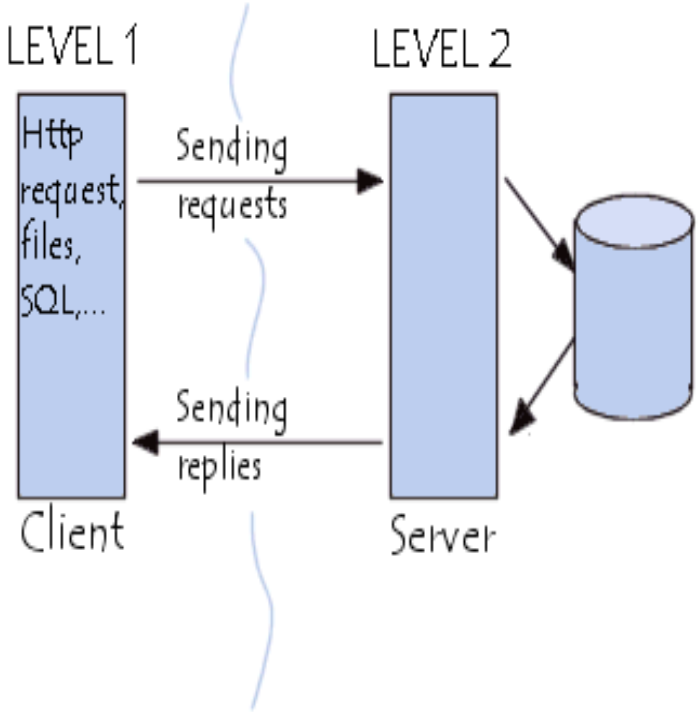
- By closing connection object statement and ResultSet will be closed automatically.
- The close() method of Connection interface is used to close the connection.
  - **public void** close()**throws** SQLException
  - Ex:
    - con.close();

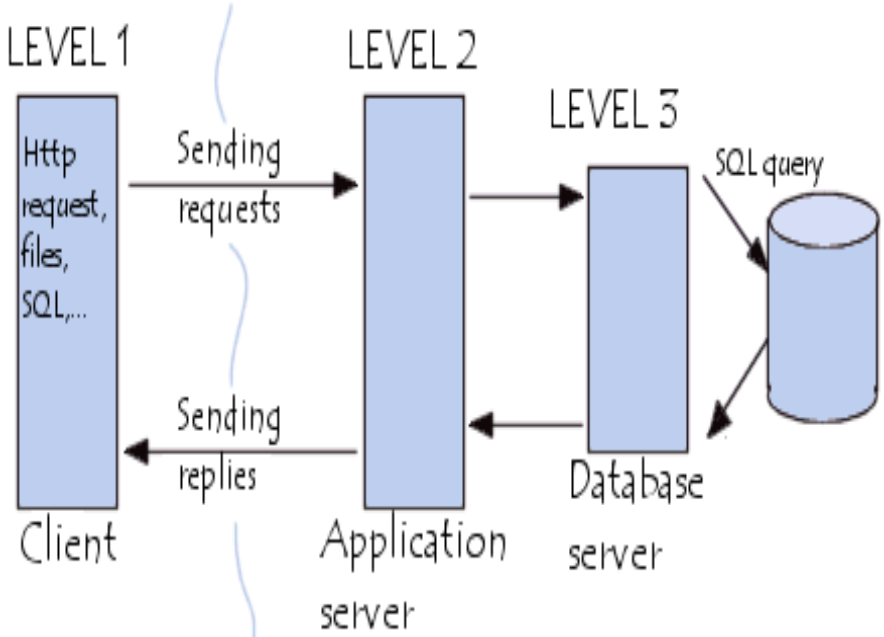
Concrete Classes for Interfaces

- ResultSet interface: oracle.jdbc.driver.OracleResultSetImpl
- Statement: oracle.jdbc.driver.T4CStatement
- Connection: oracle.jdbc.driver.T4CConnection
- PreparedStatement: oracle.jdbc.driver.T4CPreparedStatement

Types of Architecture

- Two tire architecture
- Three tire architecture





Statement Interface

- It is mainly used to execute queries.
- Few methods of Statement interface:
- public ResultSet executeQuery(String sql)
  - Used to execute select query.
- public int executeUpdate(String sql)
  - Used to execute create, insert, update, delete or drop etc
- public boolean execute(String sql)
  - Executes the given SQL statement, which may return multiple results.
- public int [] executeBatch()
  - Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts.

Statement Interface

- Insert Record:
  - Statement st = con.createStatement();
  - St.executeUpdate("insert into dept values ('comp',0130,'GPAN')");
- Delete Record:
  - Statement st = con.createStatement();
  - St.executeUpdate("delete from dept where deptno=0130");

PreparedStatement and CallableStatement Interface

- Precompiled sql statement object.
- It can read runtime input parameters.
- Ex: PreparedStatement pstmt = con.prepareStatement("update emp set salary = ? Where id = ?")
- CallableStatement interface used when Java interacting with database using stored procedures.
- Ex:
- CallableStatement cstmt = null;
- String SQL = "{call getEmpName (?, ?)}";
- cstmt = conn.prepareCall (SQL);

PreparedStatement Example

```
String sql = "update DEPARTMENTS set DEPARTMENT_NAME =
? where DEPARTMENT_ID = ?";

PreparedStatement pst = con.prepareStatement(sql);

pst.setString(1, "Information Technology");

pst.setInt(2, 60);

pst.executeUpdate();
```

Recommended Use

Interfaces	Recommended Use
Statement	Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Statement Interface Hierarchy

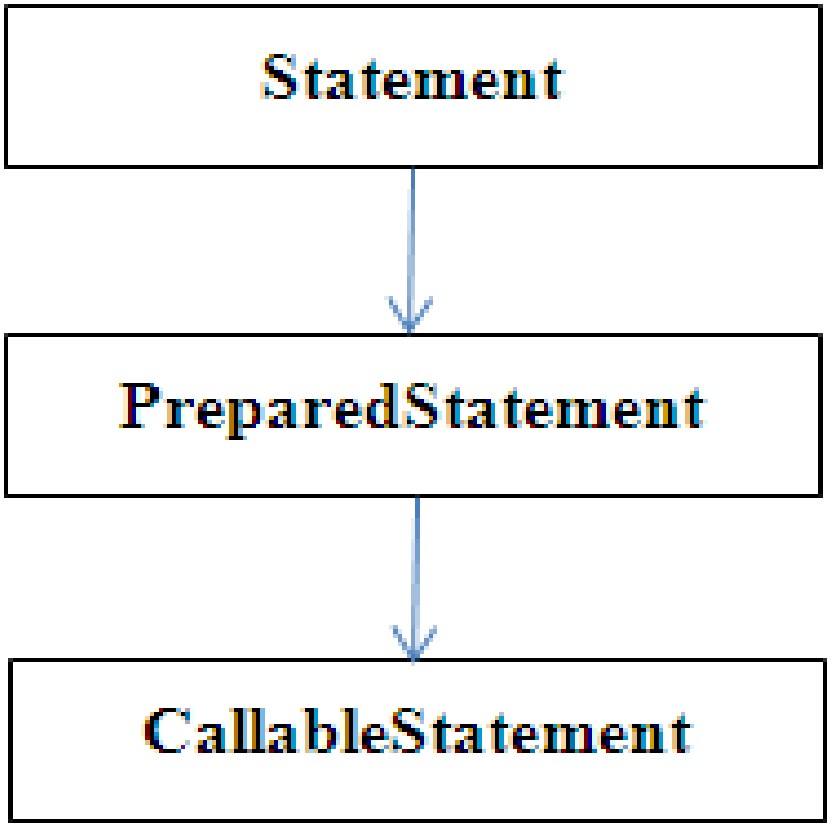


Fig: Statement interfaces hierarchy

Batch Execute Example

```
stmt.addBatch("update DEPARTMENTS set DEPARTMENT_NAME =
'Administrations' where DEPARTMENT_ID = 10");

stmt.addBatch("update DEPARTMENTS set DEPARTMENT_NAME = 'Human
Resource' where DEPARTMENT_ID = 40");

stmt.executeBatch();
```

ResultSet Interface

- ResultSet is table of data which represents a data from database.
- next() method is used to move cursor to next row.
- Type of ResultSet: Bydefault TYPE_FORWARD_ONLY
- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

ResultSet Interface methods

- beforeFirst()
- afterLast()
- first()
- last()
- previous()
- next()
- getRow()

Transaction

- All action carried out or none of them.
- For AutoCommit
  - connection.setAutoCommit(false);
- For Rollback:
  - connection.rollback();
- For Commit:
  - connection.commit();

DatabaseMetaData Interface

- Provide meta data about the database have connected to.
- For instance, you can see what tables are defined in the database, and what columns each table has, whether given features are supported etc.

DatabaseMetaData Interface

- **Obtaining a DatabaseMetaData Instance**
  - DatabaseMetaData databaseMetaData = connection.getMetaData();
- **Database Product Name and Version**
  - int majorVersion = databaseMetaData.getDatabaseMajorVersion();
  - int minorVersion = databaseMetaData.getDatabaseMinorVersion();
  - String productName = databaseMetaData.getDatabaseProductName();
  - String productVersion = databaseMetaData.getDatabaseProductVersion();

DatabaseMetaData Interface

- **Database Driver Version**
  - int driverMajorVersion = databaseMetaData.getDriverMajorVersion();
  - int driverMinorVersion = databaseMetaData.getDriverMinorVersion();
- **Listing Tables**

```
String catalog = null;
String schemaPattern = null;
String tableNamePattern = null;
String[] types = null;
ResultSet result = databaseMetaData.getTables( catalog, schemaPattern,
    tableNamePattern, types );

while(result.next()) { String tableName = result.getString(3); }
```